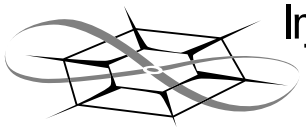
The University of Kansas



Information and Telecommunication Technology Center

Technical Report

Enhancement of Feedback Congestion Control Mechanisms by Deploying Active Congestion Control

Yoganandhini Janarthanan,
Gary Minden, and Joseph Evans

ITTC-FY2003-TR-19740-10

February 2003

Defense Advanced Research Projects Agency and the
United States Air Force Research Laboratory,
contract no. F30602-99-2-0516

Abstract

Active networking offers a change in the usual network paradigm: from passive carrier of bits to a more general computing engine. Active networking not only allows the network nodes to perform computations on the data but also allow their users to inject customized programs into the nodes of the network, that may modify, redirect or store the user data flowing through the network.

In this thesis, we focus on the benefits of active networking with respect to a problem that is unlikely to disappear in the near future: network congestion. Rather than applying congestion reduction mechanisms generically and broadly, we discuss the mechanism that allows each application to specify how losses to its data should occur in a controlled fashion. Congestion is a prime candidate for active networking, since it is specifically an intra-network event and is potentially far removed from the application. Further, the time that is required for congestion notification information to propagate back to the sender limits the speed with which an application can self-regulate to reduce congestion.

In this thesis, we propose a model for Active Congestion control, using active queue management. The SPIN verifier is used to check the correctness and completeness of the specification.

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

Active Networks

Active networking[1] offers a change in the usual network paradigm : from passive carrier of bits to a more general computing engine. In an active network, nodes can perform computations on user data as it traverses the network. Traditional data networks provide a transport mechanism to transfer bits from one end system to another, with a minimal amount of computation. In contrast, active networking not only allows the network nodes to perform computations on the data but also allow their users to inject customized programs into the nodes of the network, that may modify, redirect or store the user data flowing through the network. Moreover, active networking based solutions react faster to the changing dynamics of the network.

In an active network, customized computations are performed by the routers or switches of the network on the messages flowing through them. Routers could also interoperate with legacy routers, which transparently forward datagrams in the traditional manner. These networks are called "active" in the sense that the nodes can perform computations on, and modify, the packet contents. Further, the processing can also be customized on a per user or per application basis, in contrast to the traditional packet networks, where the routers (though they modify the packet's header) pass the user data opaquely without examination or modification.

The evolution of the active networks was triggered by the many shortcomings of the traditional networks that are in practice. The difficulty of accommodating new services in the existing architectural model, the poor performance due to redundant operations at several protocol layers and the difficulty of integrating new technologies

and standards into the shared network infrastructure are some of the problems faced by today's networks.

There are several examples that have been cited as evidence that active networking technology is either needed or already exists in some form [2]: video gateways [3], that are capable of transcoding video as it passes from one part of the network to another; multicast routers, which selectively duplicate packets before forwarding them on links; firewalls[4], which selectively filter data passing into and out of an administrative domain, to name a few, that perform user-driven computation at nodes within the network. Further, in addition to these examples, widespread implementation of active networking is likely to enable radical new applications that cannot be foreseen today.

In this thesis, we focus on the benefits of active networking with respect to a problem that is unlikely to disappear in the near future : network congestion. Rather than applying congestion reduction mechanisms generically and broadly, we discuss the mechanism that allows each application to specify how losses to its data should occur in a controlled fashion. Congestion is a prime candidate for active networking, since it is specifically an intra-network event and is potentially far removed from the application. Further, the time that is required for congestion notification information to propagate back to the sender limits the speed with which an application can self-regulate to reduce congestion.

Congestion Control Schemes

The various congestion control schemes in use are discussed in this section. The congestion control schemes can be broadly classified into source based and gateway based schemes depending on where congestion is addressed. In the source-based schemes, the end hosts observe how many packets are successfully transmitted

through the network and adjust their transmission rates accordingly. In the gateway-based schemes, the congestion control mechanism is located at the gateways.

Source based schemes

In the source-based schemes, we discuss the slow start, congestion avoidance, fast retransmit and fast recovery.

Slow start and congestion avoidance

When a sender injects multiple segments into the network, up to the window size specified by the receiver, problems can arise if there are routers and slower links between the sender and the receiver. It is possible that some intermediate router runs out of space. This is avoided using the slow start algorithm. It operates by observing that the rate at which new packets should be injected into the network is the rate at which the acknowledgements are returned by the other end. In this algorithm, whenever restarting after a loss, the congestion window is set to one packet and the minimum of the receiver's advertised window and the congestion window is sent. Congestion avoidance is a way to deal with lost packets. More details on slow start and congestion avoidance are in [5] and [6].

In these schemes, a sender is in one of the two modes: slow start or congestion avoidance. The two modes differ primarily in that the sending rate of data flow increases more aggressively in the former mode than in the latter. A sender is in slow-start mode under two conditions: (1) when it first starts transmitting data and (2) immediately after a retransmission timeout. A sender shifts from the slow-start mode to the congestion-avoidance mode at a certain threshold or immediately after a fast retransmit, which is discussed later in this paper. In the slow-start mode, the goal is to quickly fill an empty pipeline until the connection is approximately at equilibrium.

At that point, the sender shifts in to the less aggressive congestion-avoidance mode, which continues to probe the maximum network capacity. Evidently, the choice of the threshold, which is essentially an approximation of the equilibrium point of the connection, is the key to the performance of these schemes.

In practice, slow start and congestion avoidance are implemented together. To implement these congestion control and avoidance schemes, the sender keeps track of three variables: snd_wnd, cwnd, and ssthresh. Snd-wnd is the window size advertised from the receiver to sender. This advertised window size gives the sender an estimate of the available window at the receiver. Cwnd is the congestion window, and ssthresh is the slow-start threshold, which determines when a sender shifts from the slow start mode into the congestion avoidance mode.

In slow-start mode, cwnd is initialized to one segment. Each time an ACK is received, cwnd is incremented by one segment. At any point, the sender sends the minimum of snd_wnd and cwnd. Thus, cwnd reflects flow control imposed by the sender, and snd_wnd is flow control imposed by the receiver.

From the above description, we find that cwnd increases exponentially. To be more specific, assuming that the ACK's are not delayed, the sender starts by transmitting one segment and waiting for its ACK. When the ACK is received, cwnd is incremented from one to two, and two segments are sent. When each of the two segments is acknowledged, cwnd is incremented by one. The value of cwnd becomes four. This exponential pattern continues similarly until cwnd becomes greater than or equal to ssthresh. From then on, the sender shifts into the congestion avoidance mode.

A retransmission timeout or the reception of three duplicate ACKs indicates congestion. When congestion is detected, ssthresh, which dictates when the sender changes from slow-start mode to congestion-avoidance mode, is adjusted to one-half of the current window (the minimum of snd_wnd and cwnd). This makes sense in most cases, since the detection of congestion implies that the current threshold is too high. The sender is too aggressive and thus is losing packets. The threshold is lowered so that the sender shifts from exponential increase of its congestion window to

additive increase sooner in hope that the sender can slow down enough to avoid congestion in future.

Delayed acknowledgement

Here, TCP does not send an ACK the instant it receives a packet. Instead, it delays the ACK, so that if there is data going the same direction as the ACK, the ACK can be sent along, or 'piggyback' with the data. An exception is when out-of-order segments are received. In such cases, a duplicate ACK is generated immediately. Now let us discuss the effects of delayed acknowledgement, specifically in the exponential increase of slow start. In this phase, an ACK of in-order segments triggers more segments to be sent. The number of segments triggered is determined by the number of segments that were just acknowledged plus one more segment(as the congestion window is increased by one segment per ACK).

Without delayed ACK's , every ACK received acknowledges 1 segment and triggers two additional segments to be sent. However, with delayed ACK, segments are acknowledged in pairs, so every ACK for two segments received triggers three segments to be sent. Thus, the exponential expansion of the congestion window is slower when ACK's are delayed. Evidently, acknowledging every packet allows the 'pipe' to be filled more quickly. However, this aggressiveness does not automatically translate into higher throughput, because acknowledging every packet expands the congestion window faster. Such expansion leads to buffer overflow and thus packet losses. Thus, the sender must wait for retransmit timeouts, since multiple losses in one round-trip time may not be recoverable by fast retransmit.

Here, it is found that although acknowledging every packet allows the sender to open up the congestion window faster and thus to pump segments into the network faster, the sender also loses more packets as a result of aggressiveness.

Fast retransmit and fast recovery

When an out-of-order segment is received, a duplicate ACK is sent to let the other end know that a segment was received out of order, and to tell it what sequence number is expected. In fast retransmit mechanism, based on the duplicate ACK's, an educated prediction is made about which segment is missing. Without this mechanism, the sender would have to wait for a long retransmission timeout, which is on the order of seconds, before it would detect that a segment was lost. Under the fast recovery mechanism, the sender enters the congestion-avoidance instead of the slow-start mode after a fast retransmit.

If three or more duplicate ACKs are received in a row, TCP performs retransmission of what appears to be the missing segment (the segment starting with the sequence number immediately after the number acknowledged by the duplicate ACK's), without waiting for a retransmission timer to expire. Thus, TCP first lowers ssthresh to half of the window size to avoid future congestion and retransmits the missing segment. This is accomplished by adjusting the snd_nxt to that sequence number and closing down cwnd to one segment and calling tcp_output(). After the output is done, cwnd is readjusted to the highest sequence number that is outstanding so far on the connection(snd_max). This is so that the sender can continue to send new data.

After fast retransmit sends what appears to be the missing segment, congestion avoidance, but not slow start is performed. This is the fast recovery algorithm. The fast recovery mechanism refers to the way cwnd and ssthresh are adjusted so that the sender enters the congestion-avoidance mode after a fast retransmit. The fast retransmit and fast recovery algorithms are usually implemented together.

Forward Acknowledgement

The FACK congestion control algorithm addresses many of the performance problems observed in the Internet. The FACK algorithm is based on first principles of congestion control and is designed to be used with the proposed TCP SACK option.

By decoupling congestion control from other algorithms such as data recovery, it attains more precise control over the data flow in the network.

SACK, which is more of an address recovery method, helps TCP to survive multiple segment losses within a single window without incurring a retransmission timeout. When the receiver holds non-contiguous data, it sends duplicate ACKs bearing SACK options to inform the sender which segments have been correctly received. Each block of contiguous data is expressed in the SACK option using the sequence number of the first octet of data in the block and the sequence number of the octet just beyond the end of the block. In the new SACK option the first block is required to include the most recently received segment. Additional SACK blocks repeat previously sent SACK blocks, to increase robustness in the presence of lost ACKs. A SACK implementation, which uses the FACK congestion control algorithm, is referred to as 'FACK'.

The FACK algorithm uses the additional information provided by the SACK option to keep an explicit measure of the total number of bytes of data outstanding in the network. Since it accurately controls the outstanding data in the network, it is less bursty. Furthermore, since FACK uniformly adheres to basic principles of congestion control, it may be possible to produce formal mathematical models of its behavior and to support further advances in congestion control theory [7]

Gateway based schemes

The problem with the end to end congestion control schemes is that the presence of congestion is detected through the effects of congestion rather than the congestion itself. Hence it is logical to place the congestion control mechanism at the location of the congestion, i.e., the gateways. The gateway knows how congested it is and can notify sources explicitly, either by marking a congestion bit, or by dropping packets. The main drawback to marking packets with a congestion bit, as opposed to simply dropping them, is that TCP makes no provision for it currently. Floyd in [8]

states that some have proposed sending Source Quench packets as ECN messages. Source Quench messages have been criticized as consuming network bandwidth in a congested network making the problem worse. A few ways in which gateways notify the source of congestion are random early detection and explicit congestion notification.

The main goal of a congestion avoidance mechanism at the gateway is to detect incipient congestion. A congestion avoidance scheme maintains the network in a region of low delay and high throughput. The average queue size should be kept low, while fluctuations in the actual queue size should be allowed to accommodate bursty traffic and transient congestion. Because the gateway can monitor the size of the queue over time, the gateway is the appropriate agent to detect incipient congestion. Because the gateway has a unified view of the various sources contributing to this congestion, the gateway is also the appropriate agent to decide which sources to notify this congestion.

The second goal of a congestion avoidance gateway is to decide which connections to notify of congestion at the gateway. If congestion is detected before the gateway buffer is full, it is not necessary for the gateway to drop packets to notify sources of congestion. The gateway marks a packet, and notifies the source to reduce the window for that connection.

Explicit Congestion Notification

The reliance on packet drops as the indication of congestion is perfectly appropriate for a network with routers whose main function is to route packets to the appropriate output port. With the DecBit scheme discussed later in this chapter, routers detect incipient congestion by computing the average queue size, and set the ECN bit in packet headers when the average queue size exceeds a certain threshold. For networks with mechanisms for the detection of incipient congestion, the use of ECN mechanisms for the notification of congestion to the end nodes prevents

unnecessary packet drops. For bulk-data connections, the user is concerned only with the arrival time of the last packet of data, and delays of individual packets are of no concern. For some interactive traffic, however, such as telnet traffic, the user is sensitive to the delay of individual packets. For such low-bandwidth delay-sensitive TCP traffic, unnecessary packet drops and packet retransmissions can result in noticeable and unnecessary delays for the user. For some connections, these delays can be exacerbated by a coarse-granularity TCP timer that delays the source's retransmission of the packet.

A second benefit of ECN mechanisms is that with ECN, sources can be informed of congestion quickly and unambiguously, without the source having to wait for either a retransmit timer or three duplicate ACKs to infer a dropped packet. For bulk-data TCP connections, the congestion window is generally sufficiently large that the dropped packet is detected fairly promptly by the Fast retransmit procedure. But, for those cases where a dropped packet is not detected by the Fast retransmit procedure, the use of ECN mechanisms an improve a bulk-data connection's response to congestion. Some of the ECN mechanisms in TCP/IP networks are source quench messages and DecBit's ECN bit. Further discussions on ECN mechanisms are found in [8]. Floyd [8] identifies two problems with their scheme: non-compliant sources and the loss of ECN messages. The problem of a non-compliant source is a hazard for any congestion control algorithm. If there can be a source that ignores ECN messages, there could also be a source that does not respond to packet drops. However, with a congestion control scheme that uses packet drops to control congestion, any source interested in maximizing throughput cannot ignore packet drops. The author states non-compliant connections can cause problems in non-ECN environments as well as in ECN environments. With regards to ECN message loss, since the RED gateway (discussed later) continually sets ECN bits while congestion persists the loss of an ECN message will not fundamentally affect the algorithm.

One major hurdle to the application of this algorithm to TCP is the incremental deployment of ECN capable gateways and sources. One proposed

solution is to provide two bits in the header to indicate ECN compliance and the presence of congestion. This can also be done with one bit, where 'off' represents ECN capability and 'on' would represent either no ECN capability or congestion notification. When a gateway marks a packet with the bit 'off' it simply switches the bit 'on'. If the gateway wants to mark a packet with the bit 'on' it simply discards it. Notice that the one bit scheme would not work for two-way traffic where data packets travel in one direction and ACKs in the other. If a congested node sets the ECN bit for one packet, as the ACK returns to that node, it will be discarded.

ICMP Source Quench

The ICMP Source Quench is the only congestion control mechanism in the Network Layer of the TCP/IP protocol suite. Both routers and hosts play a part in the mechanism to control congestion. When a router believes itself to be congested, it sends 'Source Quench' packets back to the source of the packets causing the congestion. The Source quench ICMP tells the source to cut back the rate at which it is sending data. On receipt of these packets, the host should throttle its data rate so as to prevent router congestion. The host continues to receive source Quench ICMPs until the source is sending at an acceptable speed.

This mechanism suffers from many deficiencies in both the effectiveness of the mechanism and the specification of its implementation. Its effectiveness is flawed due to two problems: (1) it does not clearly state how a router or destination decides when it is congested, who to send a source quench message to, how often to send source quench messages and when to stop and (2) it does not clearly state how a host should react to a source quench message, by how much it should reduce its output rate, if it should inform upper layers in the network stack, and how to properly increase its output rate in the absence of source quench messages. Even worse, a router or destination may be congested and not send any source quench messages. Sources in receipt of a source quench message cannot determine if a router or

destination dropped the datagram that caused the source quench message. The source also does not know to what rate it should reduce its transmission to prevent further source quench messages.

Thus, it can be seen that this is a rudimentary congestion control method, which is loosely specified, and can lead to badly behaved operation across different implementations. It may not limit a source's transmission rate, depending on the transport protocol and it may drop packets unfairly.

The Dec Bit Mechanism

The Dec Bit mechanism, also known as the Congestion Indication mechanism, is a binary feedback congestion avoidance mechanism developed for the Digital Network Architecture at DEC and has since been specified as the congestion avoidance mechanism for the ISO TP4 and CLNP transport and network protocols.

In Dec Bit, all network packets have a single bit, the 'Congestion Experienced Bit', in their headers. Sources set this bit to zero. If the packet passes through a router that believes itself to be congested, it sets the bit to a one. Acknowledgement packets from the destination return the received Congestion Experienced Bit to the source. If the bit is set, the source knows there was some congestion along the path to the destination, and takes remedial action. In DECNET, the source adjusts its window size. In TP4, the destination alters the advertised window size, rather than returning the bit to the source.

Random drop

Random drop is a mechanism by which a router is able to randomly drop a certain fraction of its input traffic when a certain condition is true. The premise of random drop is that the probability of a randomly chosen packet belonging to a

particular connection is proportional to the connection's rate of traffic. The main appeal of random drop is that it is stateless. A router using random drop does not have to perform any further parsing of packet contents to implement the mechanism.

Random drop can be used as either a congestion recovery or a congestion avoidance mechanism. In the former, a router randomly drops packets when it becomes overloaded. In the latter, packets are randomly dropped to keep the router at its optimum power position.

Random drop depends heavily on packet sources interpreting packet loss as an indicator of network congestion. Since TCP makes this interpretation, it may be useful in the TCP environment. A problem in random drop is that it does not distinguish between 'well-behaved' and 'ill-behaved' sources. Packets are dropped for sources that are congesting a router, and for sources that are not congesting a router. Similarly, for low-rate connections such as keyboard 'telnet' connections, the loss of one packet may be a significant loss of the connection's overall traffic.

When random drop is used for congestion recovery, instead of dropping the packet that causes the input buffer to overflow, a randomly chosen packet from the buffer is dropped. Mankin [9] analyses Random Drop Congestion Recovery theoretically and as implemented in a BSD 4.3 kernel. She notes that RDCR is valuable only if the full buffer contains more packets for high-rate traffic flows than for low-rate traffic flows, and that packet arrivals for each source are uniformly distributed. In reality, Mankin notes that correlations such as packet trains and TCP window operations make the distribution non-uniform.

Floyd et. al [10] note that Drop Tail can unfairly discriminate against some traffic flows where there are phase differences between competing flows that have periodic characteristics. Their analysis shows that Random Drop can alleviate some of this discrimination, and led to the development of Random Early Detection.

Random Early Detection

Random early detection is a form of random drop used as congestion avoidance. A RED router randomly drops incoming packets when it believes that it is becoming congested, implicitly notifying the source of network congestion by the packet loss.

While the principles behind RED gateways are fairly general, and RED gateways can be useful in controlling the average queue size even in a network where the transport protocol can not be trusted to be cooperative, RED gateways are intended for a network where the transport protocol responds to congestion indications from the network. The gateway congestion control mechanism in RED gateways simplifies the congestion control job required of the transport protocol, and should be applicable to transport-layer congestion control mechanisms other than the current version of TCP, including protocols with rate-based rather than window-based flow control. However, some aspects of RED gateways are specifically targeted to TCP/IP networks. The RED gateway is designed for a network where a single marked or dropped packet is sufficient to signal the presence of congestion to the transport-layer protocol. This is different from the DecBit congestion control scheme, where the transport-layer protocol computes the fraction of arriving packets that have the congestion indication bit set.

The RED gateway calculates the average queue size, using a low-pass filter with an exponential weighted moving average. The average queue size is compared to two thresholds, a minimum threshold and a maximum threshold.

When the average queue size is less than the minimum threshold, no packets are marked. When the average queue size is greater than the maximum threshold, every arriving packet is marked. If marked packets are in fact dropped, or if all source nodes are cooperative, this ensures that the average queue size does not significantly exceed the maximum threshold. When the average queue size is between the minimum and the maximum threshold, each arriving packet is marked with probability pa, where pa is a function of the average queue size avg. Each time that a packet is marked, the probability that a packet is marked from a particular

connection is roughly proportional to that connection's share of the bandwidth at the gateway.

Thus the RED gateway has two separate algorithms. The algorithm for computing the average queue size determines the degree of burstiness that will be allowed in the gateway queue. The algorithm for calculating the packet-marking probability determines how frequently the gateway marks packets, given the current level of congestion. The goal is for the gateway to mark packets at fairly evenly spaced intervals, in order to avoid biases and to avoid global synchronization, and to mark packets sufficiently frequently to control the average queue size.

There are several significant differences between DecBit gateways and the RED gateways. The first difference concerns the method of computing the average queue size. Because the DecBit scheme chooses the last (busy + idle) cycle plus the current busy period for averaging the queue size, the queue size can sometimes be averaged over a fairly short period of time. In high-speed networks with large buffers at the gateway, it would be desirable to explicitly control the time constant for the computed average queue size; this is done in RED gateways using time-based exponential decay.

A second difference between DecBit gateways and RED gateways concerns the method for choosing connections to notify of congestion. In the DecBit scheme, there is no conceptual separation between the algorithm to detect congestion and the algorithm to set the congestion indication bit. When a packet arrives at the gateway and the computed average queue size is too high, the congestion indication bit is set in the header of that packet. Because of this method for marking packets, DecBit networks can exhibit a bias against bursty traffic; this is avoided in RED gateways by using randomization in the method for marking packets, which further avoids the global synchronization that results from many TCP connections reducing their window at the same time.

What is Active Congestion Control?

The Active Congestion Control system is used to make the feedback congestion control more responsive to network congestion, using the active networking technology. It extends the feedback congestion control system from the endpoints into the routers. Congestion is detected at the router, which also immediately begins reacting to congestion by changing the traffic that has already entered the network.

In a conventional feedback system, the congestion relief starts from the sender and moves to the congestion node, as the sender's sending rate is reduced; here, the congestion relief starts at the congestion node and the change in state that sustains that relief propagates out to the endpoint.

Feedback-based congestion control systems lack scalability with respect to network bandwidth and delay since increases in either quantity de-couples the congestion site and endpoint. The larger the end-to-end delay in a network, the longer until the endpoint can determine that the network has become congested. The higher the bandwidth of the network, the larger the amount of data the endpoint may send into a congested network in the time it takes the endpoint to detect the congestion. It has been shown that under feedback-based congestion control, the duration of congestion at the bottleneck of a connection is directly related to the bandwidth-delay product.

Active networking, with the idea of reprogramming routers with data packets, can be used to address this shortcoming of feedback control. Active Congestion Control moves the endpoint congestion control algorithms into the network where they can immediately react to congestion. The current state of the endpoint's feedback

algorithm is included in every packet. When a router experiences congestion, the router calculates the new window size that the endpoint would choose if it had instantly detected the congestion. The router then informs the endpoint of its new state. Internal network nodes beyond the congested router see the modified traffic from the router, which seems as if the endpoint had instantly reacted.

Active congestion control reduces the duration of each congestion event, and since fewer endpoints experience congestion during each congestion event, it improves the aggregate throughput. This is very effective in high-speed networks (which have a high bandwidth-delay product). Further, since fewer endpoints see congestion in ACC, and hence reduce their sending rate, the oscillations of the system as a whole is reduced.

Thesis organization

The thesis is organized as follows. Chapter 2 discusses the related work in the field of congestion control and then describes the motivation of this thesis work. Chapter 3 explains the verification of protocol models and then looks into the SPIN model checker used in this thesis work. Chapter 4 describes the difference between congestion control and congestion avoidance. It also describes the various component modules in the proposed active congestion control scheme. It further presents the finite state machine models of each component. Chapter 5 is devoted to the specification and verification of the Active Congestion Control framework using SPIN and Promela. The terminology used and the message parameters are explained here. It also explains how SPIN is used to test and verify correctness, completeness and consistency properties. The final section summarizes the work done in this thesis and discusses future work. The thesis ends with the Bibliography.

Chapter 2: Related Work

The end to end congestion control mechanisms of TCP have been a critical factor in the robustness of the Internet. However, the Internet is no longer a small, closely knit user community, and it is no longer practical to rely on all end-nodes to use end-to-end congestion control for best-effort traffic. Similarly, it is no longer possible to rely on all developers to incorporate end-to-end congestion control in their Internet applications. The network itself must now participate in controlling its own resource utilization. Sally Floyd and Kevin Fall [23] consider the potentially negative impacts of an increasing deployment of non-congestion-controlled best-effort traffic on the Internet. These negative impacts range from extreme unfairness against competing TCP traffic to the potential for congestion collapse. To promote the inclusion of end-to-end-congestion control in the design of future protocols using best-effort traffic, they argue that router mechanisms are needed to identify and restrict the bandwidth of selected high-bandwidth best-effort flows in times of congestion. They discuss several general approaches for identifying those flows suitable for bandwidth regulation. These approaches are for identifying a high-bandwidth flow in times of congestion as unresponsive, non TCP-friendly, or simply using disproportionate bandwidth. A flow that is not TCP-friendly is one whose long-term arrival rate exceeds that of any conformant TCP in the same circumstances.

Bernhard Suter, T.V.Lakshman, Dimitrios Stiliais and Abhijit Choudhury [24] present mechanisms for active buffer management that improve TCP performance in a per-flow queuing system

Samrat Bhattacharyajee et al. [22] have considered a range of schemes for processing application data during congestion, including unit level dropping, media transformation and multi-stream interaction. They have also presented some architectural considerations for a simple approach, in which packets are labeled to

indicate permitted manipulations. Their results suggest that congestion control makes a good case for active networking, enabling schemes that are not possible within the conventional view of the network.

In the University of Southern California, Ted Faber has simulated studies of an active congestion control system based on TCP congestion control mechanisms [21]. The active system and the standard TCP congestion control in networks with and without bursty cross traffic have been simulated and compared. They have shown that when bursty cross-traffic is added, the active system shows as much as an 18% throughput improvement. All the simulations were made using ns, a simulator produced by the University of California Berkeley, the Lawrence Berkeley National Labs, and the Virtual InterNet project. They have extended the simulator to implement the ACC algorithms in the routers and endpoints, but not to allow full AN programmability. The modified simulator does not compile or interpret code from simulated packets.

Our Approach - Motivation

There are several issues that have motivated this approach. The first one is that active networking, in its most general form, requires substantial changes in network architecture. To move the network in the direction of these changes, active networking must offer some benefits, assuming that functionality will not be added to end systems unless there is some benefit in doing so. Also, switch manufacturers and network operators will not upgrade their switches to support active networking unless there is ultimately some benefit to their customers.

Secondly, both computational power and transmission bandwidth will continue to increase, but so will the application requirements for bandwidth. In particular, we expect that network node congestion will be due to bandwidth limitations and that the

congested switches will still have considerable processing power, as compared to buffering, available.

Finally, there will always be applications that prefer to adapt their behavior dynamically, rather than reserving bandwidth in advance, in order to match the available network bandwidth. This is based on several observations: (1) there will be times when the network rejects requests for bandwidth and the applications will have no choice. (2) The reserved bandwidth is likely to cost more. (3) The sending application's ability to trade processing for transmission bandwidth in reaction to congestion in the network increases as the computing speeds increase.

We also know that the sender-adaptation model [6] that has worked well in the Internet, presents a couple of well-known challenges. The first one is the time interval required for the sender to detect congestion and adapt in order to bring losses under control and have the controlled-loss data propagate to the receiver. During this interval, the receiver experiences uncontrolled loss, resulting in a reduction in quality of service that magnifies the actual bandwidth reduction. As transmission and application bandwidths increase, this problem is exacerbated because propagation delays remain constant.

Another challenge of sender adaptation is detecting an increase in available bandwidth. This problem, which is worse for continuous-media applications, arises in best-effort networks because loss is the only mechanism for determining available bandwidth. Hence, if a sender adapts to congestion by changing to a lossier encoding, it must detect the easing of congestion by periodically reducing compression and waiting for feedback from the receiver. In the case of long-lived congestion, this dooms the receiver to periodic episodes of uncontrolled loss.

Hence, we conclude that a useful application of active networking is to move those adaptations a sender might make into the network itself, in order to solve the above problems. Our aim is to ensure that, as far as possible, losses occur in a controlled and application-specific manner.

In this thesis, we have verified the active congestion control framework. Verification is an extremely useful technique for early detection of design errors and also complements the design documentation. Verification also has the advantage that it forces the designer to reproduce all design decisions and thus also helps in finding logical errors.

Summary

In this chapter, we have discussed the related work and the motivation for this thesis work. In the next chapter, the various approaches for specification and verification of protocols have been discussed.

Chapter 3: Specification and Verification

Introduction

Active Networking enables users to customize network processing through the deployment of application-specific protocol frameworks into the nodes of the network. Injecting user-defined code into the network raises important issues regarding safety of the active nodes and the network in general. The performance and security of the network is compromised if the injected code contains inadvertent mistakes or if the protocol framework does not work as expected. A major requirement in such systems is to enable developers to construct protocol frameworks that operate reliably.

Using formal methods of specification and verification increases the confidence of user-defined protocol frameworks. Specification is the process of describing a system and its properties. Formal specification uses a language with mathematically defined syntax and semantics. Properties described by the specification can include functional behavior, timing behavior, performance characteristics or internal structure. Verification is the process of mathematically proving the veracity of the specification.

Formal specifications are simultaneously precise, concise and clear. They can be mechanically checked for both syntax and certain semantic "goodness" properties, helping designers to catch mistakes. Use of formal methods does not guarantee correctness. However, they can greatly increase confidence in the system by revealing inconsistencies, ambiguities and incompleteness that would have otherwise gone undetected.

Protocol Verification

There has been a number of studies on protocol verification that deal with the issue of correctness of a given protocol specification by testing it for safety and liveness properties. Verification of safety properties guarantees that the protocol does not violate any constraints imposed or the system always ends in one of the valid end-states determined by the designer. Verification of liveness properties tests that the protocol does not deadlock and that it always makes progress.

For the last two decades, verification techniques have been applied successfully in software and hardware engineering, especially in the communications protocol domain. Various techniques have been proposed, ranging from pure simulation to model checking. The widely used simulation techniques cannot cover all design errors, especially for large systems. Like testing techniques, they are used to detect errors, but not to prove the correctness of the design. During the past decade, model-checking techniques have established themselves as significant means for design validation. A given design is validated against specific and general properties.

There are two major approaches to verification of systems: model checking and theorem proving. In theorem proving, the proof that the design realizes the stated behavior is mechanically checked by a theorem prover. Theorem proving based verification efforts are known to be highly interactive. Model checking, on the other hand, can be fully automated. In model checking, a set of desired properties of a model of the design is stated in some form of logic and verified using a model checker for that logic. Theorem provers operate at a high level that take axioms, pre-conditions and generate proves. The disadvantage is that they require skilled intervention. Model checkers are more accessible tools. You provide a model, the model checker then tries everything the model can do and reports deadlocks if it finds any.

Theorem Proving

Theorem proving is a technique where both the system and its desired properties are described in terms of algebraic or logical formulae. The logic is given by a formal system, which defines a set of axioms and a set of inference rules. Theorem proving is the process of finding the proof of a property from the axioms of the system. Theorem provers rely on techniques like structural induction, rewrite-rules and proofs by contradiction to prove properties of systems. But finding proofs in theorem proving systems is a difficult process.

Communication systems have the notion of state, which has to be embedded in the system model, for which theorem proving systems are not well equipped. Further, theorem proving systems also require that the description of the system be abstracted so that the properties can be clearly specified. While useful for verification of the properties, a consequence of this strategy is that the implementation differs substantially form the specification. This makes it difficult to ascertain if the implementation preserves the properties expressed by the specification.

Model Checking

Model checking is a technique that relies on building a finite model of a system and checking that the desired property holds in that model. Generally, the check is performed as an exhaustive state space search that is guaranteed to terminate since the model is finite-space. In contrast to theorem proving, model checking is automatic and fast [13]. Model checking can be used to check partial specifications, and so it can provide useful information about a system's correctness even if the system has not been completely specified.

Two different fields of model checking have arisen: formal verification of software protocols and software systems, like SPIN [12] and formal verification of digital hardware.

The very first two temporal logic model checkers were EMC [13] and CAESAR. The SPIN system [18], [28], that has been used in this thesis, uses partial order reduction to reduce the state explosion problem [16]. There are other checkers, such as the behavior conformance checkers and combination checkers, roughly classified based on whether the specification they check is given as a logical formula or as a machine to eliminate unnecessary states from a system model.

The main disadvantage of model checking is the state explosion problem. If the number of states is too large, the model checker requires unreasonable amount of time and memory to complete verification - this is known as the state-space explosion problem. In 1987, McMillan used Bryant's ordered binary decision diagrams (BDDs)[14] to represent state transition systems efficiently, thereby increasing the size of the systems that could be verified. Other promising approaches to alleviating state explosion include the exploitation of partial order information [16], localization reduction [15] and semantic minimization [17] to eliminate unnecessary states from a system model. Thus, it is apparent that the advantages of using a model checking system for verifying communication protocol frameworks far outweigh its limitations.

SPIN Model Checker

SPIN is a generic model checking system that supports the design and verification of asynchronous process systems. SPIN verification models are proving the correctness of process interactions. Process interactions are specified using rendezvous and buffered message passing through channels and/or through access to shared variables. SPIN provides an intuitive, program-like input language called Promela [18, Gerald Holzmann 1997] for specifying design choices without implementation detail. It provides a powerful, concise notation for expressing general correctness requirements

and a methodology for establishing the logical consistency of the design choices using Promela and the matching correctness requirements.

The typical model for working is to start with the specification of a high level model of a concurrent system, or distributed algorithm, typically using SPIN's graphical front-end XSPIN. After fixing syntax errors, interactive simulation is performed until basic confidence is gained that the design behaves as intended. Then, in a third step, SPIN is used to generate an optimized on-the-fly verification program from the high level specification. This verifier is compiled, with possible compile-time choices for the types of reduction algorithms to be used, and executed. If any counterexamples to the correctness claims are detected, these can be fed back into the interactive simulator and inspected in detail to establish, and remove, their cause.

The easiest way to get started with SPIN is to use the graphical interface Xspin. The graphical interface runs independently from SPIN itself, and helps by generating the proper SPIN commands based on menu selections. Xspin runs SPIN in the background to obtain the desired output, and wherever possible, it will attempt to generate a graphical representation of such output. Xspin knows when and how to compile code for the model checkers that SPIN can generate, and it knows when and how to execute it, so there is less to remember.

The description of a concurrent system in PROMELA consists of one or more user-defined process templates, or proctype definitions, and at least one process instantiation. The templates define the behavior of different types of process. Any running process can instantiate further asynchronous processes, using the process templates. SPIN translates each process template into a finite automaton. The global behavior of the concurrent system is obtained by computing an asynchronous interleaving product of automata, one automaton per asynchronous process behavior. The resulting global system behavior is itself again represented by an automaton. This interleaving product is often referred to as the state space of the system, and because it can easily be represented a s a graph, it is also commonly referred to as the global reachability graph.

SPIN performs the verification by taking the correctness claim that is specified as a temporal logic formula, converts that formula into a Buchi automaton, and computes the synchronous product of this claim and the automaton representing the global state space. The correctness claims ate used to formalize system behaviors that are undesirable and the verification process then either proves that such behaviors are impossible or it provides detailed examples of behaviors that match. SPIN's verification procedure is based on an optimized depth-first graph traversal method. The cycle detection method used in SPIN is of central importance. The method is required to be compatible with all modes of verification, including exhaustive search, bit-state hashing, and partial order reduction techniques.

Partial Order Reduction Method

SPIN uses a partial order reduction method [16, Peled 1994] to reduce the number of reachable states that must be explored to complete a verification. The reduction is based on the observation that the validity of an LTL formula is often insensitive to the order in which concurrent and independently executed events are interleaved in the depth-first search. Instead of generating an exhaustive state space that includes all execution sequences as paths, the verifier can generate a reduced state space, with only representatives of classes of execution sequences that are indistinguishable for a given correctness property. The implementation of this reduction method is based on a static reduction technique, described in [19, Holzmann and Peled, 1994], that, before the actual verification begins, identifies cases where partial order reduction rules can safely be applied when the verification itself is performed. This static reduction method avoids the runtime overhead that has plagued partial order reduction strategies in the past.

Supertrace Verification

To conserve memory while verifying models, SPIN's supertrace or BitState verification technique is used as opposed to exhaustive search. This technique is used for large problem sizes that preclude exhaustive verification. It enables a high-coverage approximation of the results of an exhaustive run that can be performed in relatively small amounts of memory. The algorithm uses 2 bits of memory to store a reachable state. The bit addresses are computed with two statistically independent hash functions. If storing one reachable system state requires S bytes of memory, and if the machine has M bytes of memory available, the model checker exhausts its available memory after generating M/S states. If the true number of reachable states, R, exceeds M/S, then the problem coverage of that verification run is M/(RxS). Under the same system constraints, the bit state hashing technique can produce an average problem coverage close to 1 (that is, approximately 100% coverage). In general, when M < RxS, the supertrace technique typically realizes a far superior problem coverage than standard exhaustive searches [20, Holzmann, June 1995].

Summary

In this chapter, the various approaches for specification and verification of protocols have been presented and it has been found that the model-checking approach is the most suitable for our framework. In the next chapter, we describe the composition and verification of active congestion control framework.

Chapter 4: Active Congestion Control Framework

Current networks require system-wide deployment whenever a new protocol is developed and is ready to be introduced in the network. But the current network infrastructure is rigid and fixed in the sense that developing and introducing new protocols in the network requires a time-consuming standardization process. Active networking provides a new paradigm in which the nodes of the network are programmable; that is, they provide an execution platform on which user code can be executed. Applications can customize network resources for dynamic adaptation by injecting the code in the network that is executed at the network nodes. In this chapter, we explain the Active Congestion Control Framework and its individual components. We also compare the concept of congestion avoidance with that of congestion control. Later, we look at the terminology used to describe the modules in the framework and the Finite State Machine Model of each module in detail.

Congestion Avoidance and Congestion Control

Congestion is a significant problem in computer networks today, due to increasing use of the networks, as well as due to increasing mismatch in link speeds caused by intermixing of old and new technology. Recent advances in technology have resulted in a significant increase in the bandwidths of computer network links. This causes a heterogeneity, as the new technologies much coexist with the old low bandwidth media and this results in mismatch of arrival and service rates in the intermediate nodes in the network, causing increased queuing and congestion.

Now, let us look at the difference between congestion avoidance and congestion control. Traditional congestion control schemes help improve the performance after congestion has occurred. Figure 1 shows the general patterns of throughput and response time of a network as the network load increases.

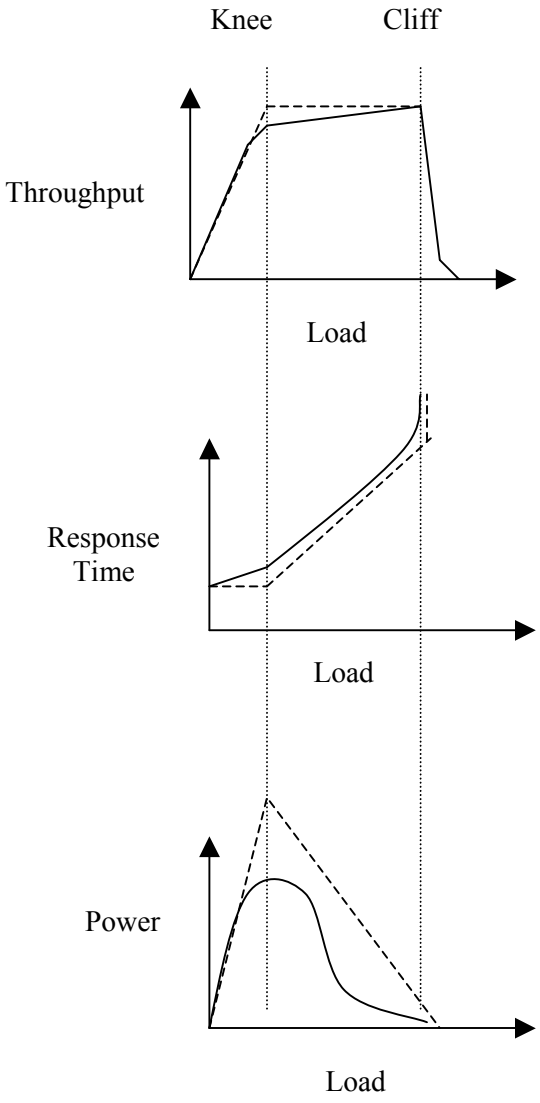


Figure 1: Response Time , throughput and power as a function of network load

If the load is small, throughput keeps up with the load. As the load increases, throughput increases. After the load reaches the network capacity, throughput stops increasing. If the load is increased any further, the queues start building, potentially resulting in packets being dropped. When the load is increased beyond this point, throughput drops suddenly and the network is said to be congested. The response time also follows a similar pattern. At first, the response time increases little with load. As the queues start building up, the response time increases linearly until finally, as the queues start overflowing, the response time increases drastically. The point at which throughput approaches zero is called the point of congestion collapse. At this point, the response time approaches infinity. The purpose of a congestion control scheme is to detect the fact that the network has reached the point of congestion collapse resulting in packet losses, and to reduce the load so that the network returns to an uncongested state.

The point of congestion collapse is called a cliff, as the throughput falls off rapidly after this point. The term knee is used to describe the point after which the increase in the throughput is small, but after which a significant increase in the response time results.

A scheme that allows the network to operate at the knee is called a congestion avoidance scheme, as distinguished from a congestion control scheme that tries to keep the network operating in the zone to the left of the cliff. A properly designed congestion avoidance scheme will ensure that the users are encouraged to increase their traffic load as long as this does not significantly affect the response time and are required to decrease the load if that happens. A congestion avoidance scheme allows a network to operate in the region of low delay and high throughput. These schemes prevent a network from entering the congested state in which the packets are lost. Congestion control schemes are still required, however, to protect the network should it reach the cliff due to transient changes in the network.

Thus, congestion control is a recovery mechanism, while congestion avoidance is a prevention mechanism. In other words, congestion control procedures are curative and the avoidance procedures are preventive in nature. The point at which a congestion control scheme is called upon depends upon the amount of memory available in the routers, whereas the point at which a congestion avoidance scheme is invoked is independent of the memory size.

Mechanisms in Congestion avoidance/control

Congestion control and congestion avoidance are dynamic system control issues. They have two parts like all other control schemes - a feedback mechanism and a control mechanism.

Feedback mechanism: Allows the network to inform its users (sources or destinations) of the current state of the system.

Control mechanism: Allows the users to adjust their loads on the system. In our model, this mechanism is present in the active router.

The feedback mechanism has the following alternatives:

1. Choke packet, where congestion feedback via packets sent from routers to sources.
2. Feedback included in the routing messages exchanged among routers.
3. End-to-end probe packet sent by sources.
4. Reverse feedback, where each packet containing a congestion feedback field filled in by routers in packets is going in the reverse direction.
5. Forward feedback, where each packet containing a congestion feedback field filled in by routers in packets is going in the forward direction.

In the next section, the concept of active congestion control and how it has been modeled is explained.

Active Congestion Control

The Active Congestion Control (ACC) system is used to make the feedback congestion control more responsive to network congestion, using the active networking technology. ACC extends the feedback congestion control system from the endpoints into the routers. Congestion is detected at the router, which also immediately begins reacting to congestion by changing the traffic that has already entered the network.

In a conventional feedback system, the congestion relief starts from the sender and moves to the congestion node, as the endpoint's sending rate is reduced; here, the congestion relief starts at the congestion node and the change in state that sustains that relief propagates out to the endpoint.

Feedback-based congestion control systems lack scalability with respect to network bandwidth and delay since increases in either quantity de-couples the congestion site and endpoint. The larger the end-to-end delay in a network, the longer until the endpoint can determine that the network has become congested. The higher the bandwidth of the network, the larger the amount of data the endpoint may send into a congested network in the time it takes the endpoint to detect the congestion.

Active networking, with the idea of reprogramming routers with data packets, can be used to address this shortcoming of feedback control. Active Congestion Control moves the endpoint congestion control algorithms into the network where they can immediately react to congestion. The current state of the sender's feedback algorithm is included in every packet. When a router experiences congestion, the router calculates the new window size that the sender would choose if it had instantly detected the congestion. The router then informs the endpoint of its new state.

Internal network nodes beyond the congested router see the modified traffic from the router, which seems as if the endpoint had instantly reacted.

Active congestion control reduces the duration of each congestion event, and since fewer endpoints experience congestion during each congestion event, it improves the aggregate throughput. This is very effective in high-speed networks (which have a high bandwidth-delay product). Further, since fewer endpoints see congestion in ACC, and hence reduce their sending rate, the oscillations of the system as a whole is reduced.

Active Congestion Control Framework

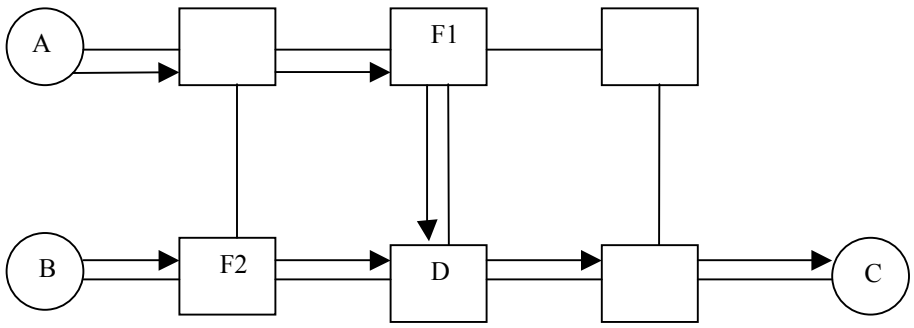


Figure 2: An active congestion control network during congestion

Consider the situation depicted in Figure 2. The packet streams from A and B pass through a router D, on the way to the destination C, congesting D. The flow of the packet streams is depicted by the arrows in the figure. In the conventional feedback system, A or B will detect congestion, either when they receive notification from the congested router, or when they deduce the existence of congestion due to packet loss or excessive delays. By the time A has realized that D is congested, it has spent at least the propagation delay from A to D and back, sending packets as though the network were uncongested, thereby making the congestion worse.

This delay is removed under ACC. Router D has been programmed by the first packet of the connection with instructions on how to react to congestion, and subsequent packets include information on the current state of the endpoint's congestion control algorithm. When D detects the congestion, it decides what action the endpoint would take if it had detected congestion in the state reflected by its most recent packet. The router then installs filters that either delete packets that the source would not have sent or perform some action on the packets, as specified by the user, provided the resources are allowed. These filters may be installed at the congested router's interfaces or at those of neighboring routers (F1 or F2). Finally, the congested router sends a message to the sender telling it the new state of its congestion control system.

The formats of the messages in ACC are discussed in detail in the subsequent chapters.

Router Algorithms

Considerable research has been done on Internet dynamics and it has become clear that TCP congestion avoidance mechanisms, while necessary and powerful, are not sufficient to provide good service in all circumstances. Basically, there is a limit to how much control can be accomplished from the edges of the network. Some mechanisms are needed in the routers to complement the endpoint congestion avoidance mechanisms.

The router algorithms related to congestion control can be classified into two classes - queue management and scheduling algorithms. Queue management algorithms manage the length of packet queues by dropping packets when necessary or

appropriate, while scheduling algorithms determine which packet to send next and are used primarily to manage the allocation of bandwidth among flows.

Active Queue Management

The traditional technique for managing router queue lengths is to set a maximum length for each queue, accept packets for the queue until the maximum length is reached, then reject subsequent incoming packets until the queue decreases because a packet from the queue has been transmitted. This technique is known as tail drop, since the packet that arrived most recently is dropped when the queue is full. This method has the disadvantage of lockout, wherein a single or few connections monopolize queue space. Another drawback is that it allows queues to maintain a full status for long periods of time, since tail drop signals congestion only when the queue has become full. These drawbacks are overcome using RED, an active queue management discussed in the Introduction chapter. The advantages of active queue management for responsive flows are summarized below:

- Reduction of the number of packets dropped in routers

By keeping the average queue size small, active queue management provides greater capacity to absorb bursts without dropping packets. It is noted that while RED can manage queue lengths and reduce end-to-end latency even in the absence of end-to-end congestion control, RED will be able to reduce packet dropping only in an environment that continues to be dominated by end-to-end congestion control.

- Lower-delay interactive service

By keeping the average queue size small, queue management reduces the delays seen by flows. This is particularly important for interactive applications such as short web transfers, telnet traffic, or interactive audio-video sessions, whose performance is better when the end-to-end delay is low.

- Lock-out behavior avoidance

Active queue management prevents the lockout behavior by ensuring that there will almost always be a buffer available for an incoming packet.

Policies of a router

There is a range of policies that a router might use to identify which high-bandwidth flows to regulate. For a router with active queue management such as RED [10], the arrival rates of high-bandwidth flows can be efficiently estimated from the recent packet drop history at the router. Because the RED packet drop history constitutes a random sampling of the arriving packets, a flow with a significant fraction of the dropped packets is likely to have a correspondingly significant fraction of the arriving packets. Thus, for higher bandwidth flows, a flow's fraction of the dropped packets can be used to estimate that flow's fraction of the arriving packets. The policies for regulating high-bandwidth flows range in the degree of caution. One policy would be only to regulate high-bandwidth flows in times of congestion when they are known to be violating the expectations of end-to-end congestion control, by being either unresponsive to congestion or exceeding the bandwidth used by any conformant TCP flow under the same circumstances. In this case, an unresponsive flow could either be restricted to the same bandwidth as a responsive flow (the more cautions approach), or could be given less bandwidth than a responsive flow (the more powerful but less cautious approach).

Another observation to be considered is that TCP congestion avoidance is not sufficient to provide good service in all circumstances and that, because of limitations on what can be accomplished purely on an end-to-end basis, mechanisms are needed in routers to enhance end-to-end control. The RED-manifesto suggests using active queue management and more specifically the RED packet dropping scheme to prevent lockout, where one connection monopolizes the link and to prevent global synchronization of windows, which can happen in a FIFO buffer when a burst of packets arrive to a full buffer due to packet drops from all flows.

Some issues of significance in the active router are the decision frequency and the decrease algorithm.

- Decision Frequency

The decision frequency component decides how often to change the window. Changing it too often leads to unnecessary oscillations, whereas changing it infrequently leads to a system that takes too long to adapt. Based on the system control theory, the optimal control frequency depends on the feedback delay, that is, the time between applying a control and getting feedback from the network corresponding to this control.

In networks, it takes one round trip delay for the new window size to take effect and another round trip delay for the resulting change to be fed back from the network to the users. This has generally restricted the window size adjustment to have at least two round trip delays.

- Decrease Algorithm

The purpose of this algorithm is to determine the amount by which the window should be changed once a decision has been made to adjust it. The decrease is generally a function of the past history, that is, the window used in the last cycle. Hence, the state information in the packets plays a crucial role in the decrease algorithm, and the congestion control/avoidance scheme. The decrease algorithm is governed by the following goals:

Efficiency	: The system bottleneck should be operating at the knee.
Fairness	: The users sharing a common bottleneck should get a fair ratio of throughput.
Minimum convergence time	: The network should reach the optimal (fair as well as efficient) state as soon as possible, starting from any state.

Minimum Oscillation size : Once at the optimal state, the user windows oscillate continuously below and above this state. The algorithm should be such that the oscillation size is minimal.

Terminology

The terms that are used to describe the Composition of the Active Congestion Control Framework are described below:

Component

A component is defined as an entity that implements a piece of functionality [26]. It can be combined with other components to form protocols.

Active router

Any router in the active network, having the active functionality is defined as an active router.

Service

This is the functionality available in the network, through the use of one or more of the components.

Active Host/User

An entity which can interact with the active router and which uses the service provided by the active router.

In the next section we look at the various components in Active Congestion Control System.

Components in Active Congestion Control System

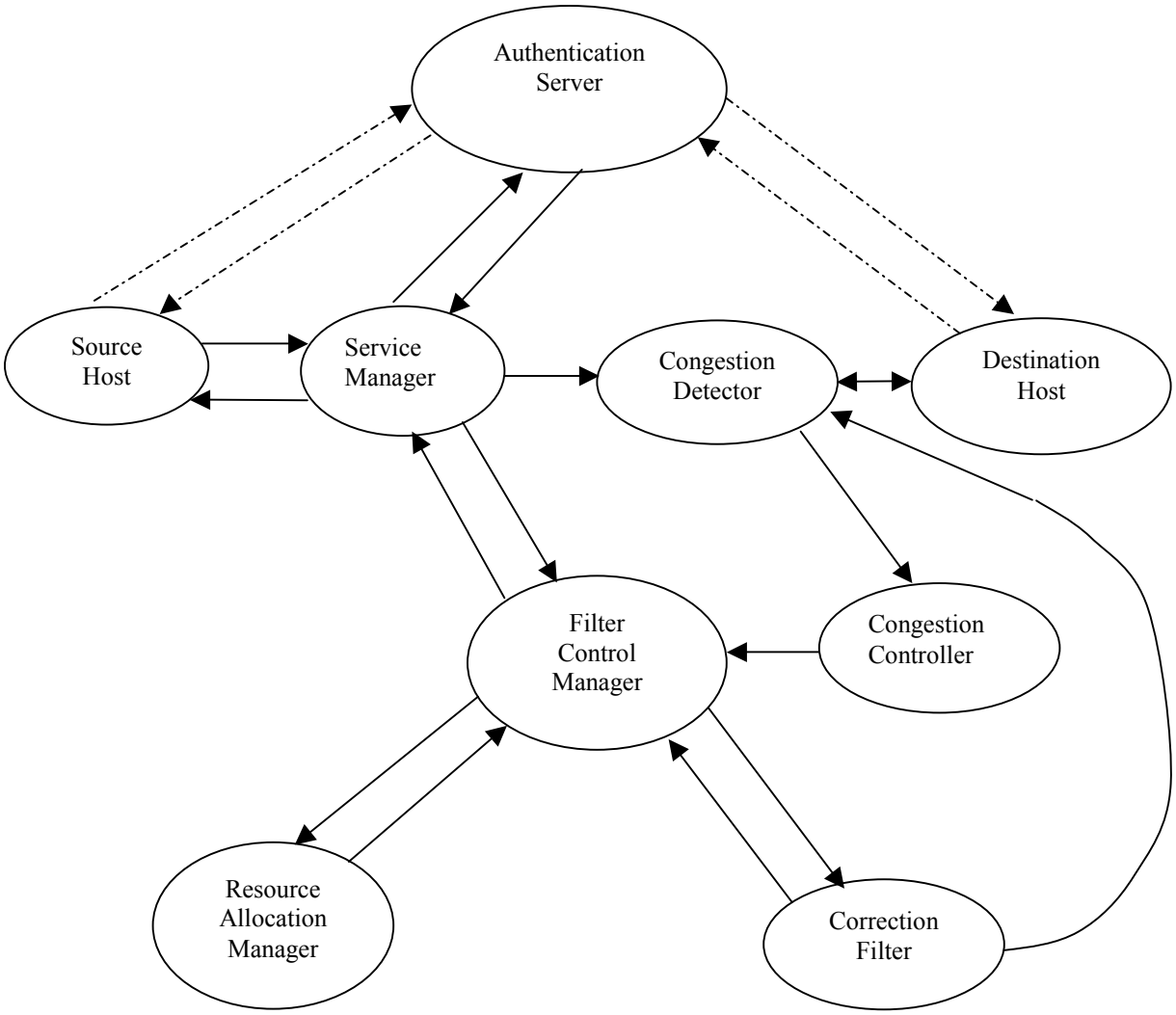


Figure 3: Components of the Active Congestion Control Framework

Figure 3 shows the various components of the Active Congestion Control Framework and their interaction. The components that have been identified in the ACC system are listed below:

Active Host

This sends the message and reacts to the feedback from the active router. In our model, the hosts have two levels of authentication: (1) The hosts can either trust the Service manager, in which case the hosts don't invoke the Authentication server, or (2) The hosts don't trust anyone and invoke the Authentication server to first check the authenticity of the Service manager.

Service Manager

This receives the message from the hosts and then sends the packet to the authentication server for verification. If the host is identified to be authentic, it transfers the packet to either the Filter Control Manager or the RED router (Congestion Detector), depending on whether it is a control message type or data message type. It also directs the Filter Control Manager to uninstall the filter. This happens in two scenarios. The service manager has timeouts for each flow. When a flow is inactive for long, it sends an uninstall message to the Filter Control Manager (FCM). It also sends an uninstall message to the FCM when an explicit request is made by a sender to change the filter.

Authentication Server

The authentication server is invoked by the Service Manager to verify the authenticity of the sender. Further, the hosts also invoke it, if the hosts look for extra authentication and don't trust the service manager. In our design, we have made the provision for both the situations, that is, when the host trusts the service manager and

when it doesn't, called the non-trusting hosts. The non-trusting source verifies the authenticity of the service manager. Authentication is done in three stages:

(1) Source Authentication, where it is verified that the source of the message is indeed what is mentioned in the packet.
(2) Integrity Check, where the integrity of the message is checked, thereby verifying that the message is not modified in transit.
(3) Source Reliability, where the source is checked for misbehaviors in the past.

The authentication server either uses public key cryptography or secret key cryptography for authentication. One way to do provide security is to use a trusted node known as a Key Distribution Center (KDC), which uses secret keys. The KDC knows the keys for all the nodes in the networks. If a new node is installed in the network, only that new node and the KDC need to be configured with a key for that node. The public key cryptography based scheme has a trusted node known as a Certification Authority (CA), that generates certificates, which are signed messages for each node or entity and the corresponding public key. All the nodes will need to be preconfigured with the Certification Authority's public key so that they can verify its signature on certificates. Certificates can be stored in any convenient location, such as the directory service, or each node can store its own certificate and furnish it as part of the authentication exchange. To scale the authentication schemes further up, we can use multiple KDC and CA domains, where the world is broken into domains and each domain has one trusted administration.

The authentication server also provides message integrity. In the public key based method, the sender computes the hash of the message (using MD5, maybe) and then signs the message digest using her private key, since computing a message digest is faster than public key operations and since the message digest is usually a smaller quantity to sign than the message. In the secret key based method, the Message Integrity Check is calculated, which is the encrypted message digest, where the

message digest is encrypted with the shared secret key between the sender and the Central authority.

Finally, the authentication server makes sure that the sender hasn't misbehaved in the past, by checking a database that it maintains, containing the misbehaved hosts. This list contains hosts that either tried to hog the resources or tried to act as imposters.

Once the three stages of authentication is successful, the authentication server sends a message back to the requesting component, stating the success of the request. If the request fails at any of the stages, it sends a failure status, with details about the stage in which the failure occurred.

Congestion Detector

This is a sub-component of the router. Data packets arrive here after the sender is verified to be authentic. Two of the possible types are

- *Drop tail router*

Here, the parameter that determines congestion is the *buffer size*. When the buffer is full, the router starts dropping the packets.

- *RED router*

The RED router discards packets before its queue is full. It picks a random packet to discard. The probability that an arriving packet is marked for discard is proportional to the amount that the router's current queue length exceeds a *threshold*. In our model, we have modified the RED router to send the packets to the congestion controller, instead of discarding it. The congestion controller's operation is discussed in the next sub-section. Thus, the router provides the functionality of a RED router, with additional enhancements instead of dropping the packets, and it provides congestion avoidance.

The RED router calculates the average queue size, using a low-pass filter with an exponential weighted moving average. The average queue size is compared to two thresholds, a minimum threshold and a maximum threshold.

When the average queue size is less than the minimum threshold, no packets are marked. When the average queue size is greater than the maximum threshold, every arriving packet is marked. If marked packets are in fact dropped, or if all source nodes are cooperative, this ensures that the average queue size does not significantly exceed the maximum threshold. When the average queue size is between the minimum and the maximum threshold, each arriving packet is marked with probability p_a, where p_a is a function of the average queue size average. Each time that a packet is marked, the probability that a packet is marked from a particular connection is roughly proportional to that connection's share of the bandwidth at the gateway. The general RED gateway algorithm is

For each packet arrival

Calculate the average queue size avg.

If min_{th} <= avg. < max_{th}

Calculate probability p_a

With probability p_a:

Mark the arriving packet

Else if max_{th} <= avg.

Mark the arriving packet.

Thus the RED gateway has two separate algorithms. The algorithm for computing the average queue size determines the degree of burstiness that will be allowed in the gateway queue. The average queue size is calculated every time a packet arrives or periodically after an interval. It is found as follows, using an exponential weighted moving average, where w_q is the weight associated with the current queue size:

Average queue size avg_q = (1 - w_q) * prev_avg_q + w_q * current_q_size

The optimum values of min_{th} and max_{th} depend on the desired average queue size. The RED gateway functions most effectively when max_{th} - min_{th} is larger than the typical increase in the calculated average queue size in one roundtrip time.

The algorithm for calculating the packet-marking probability determines how frequently the gateway marks packets, given the current level of congestion. The packet-marking probability, $p_{b,}$ is calculated as

$$p_b = max_p * (avg\_q - min_{th})/(max_{th} - min_{th})$$

where = max_p is the maximum probability for a packet to be dropped. The goal is for the gateway to mark packets at fairly evenly spaced intervals, in order to avoid biases and to avoid global synchronization, and to mark packets sufficiently frequently to control the average queue size. The marked packets are then forwarded to the congestion controller.

Congestion Controller

This is also present at the router - this performs the following operations:

1. Calculates the correct window size for the source from the state information in the current packet (Using TCP's window adjustment algorithm, the new window is half the old).
2. Sends a packet with the new window size to the source and
3. Forwards the packet to the Correction Filter, through the Filter Control Manager.

Filter Control Manager

The Filter Control Manager contacts the Resource Allocation Manager, to make sure that the resources required are available and are not exploited by a single user. It then installs a filter at the active router's interface. It also takes care of uninstalling a filter, which occurs either because a particular session is idle for too long and a time-out occurs or due to explicit request from the sender, requesting a change in the type of filter installed. There is also the option of installing the filter in any router in the path

between the sender and the current active router. In this case, the filter control manager locates the cooperating routers in the process of filter installation, with the help of the resource allocation manager.

Resource Allocation Manager

This module makes sure that the resources are available for the particular request. It also makes sure that any host doesn't hog the resources. It also takes care of pre-emption of the allocated resources, depending on the priority of the new request. It takes into consideration the number of requests from a host, the requested processor and memory allocation requirements, whenever a resource allocation request is made. It is the task of the resource allocation manager to identify the nodes that allow the installation of the filter, in case the filter is installed in a router along the path between the sender and the active router.

Correction Filter

The default filter, which the most rudimentary form of the filter, deletes all packets from that endpoint until it begins to act on the router's feedback. Since TCP responds to only one packet drop per round trip time, packets dropped by the filters will not cause endpoints to close their windows more quickly than they would, in the face of a single packet drop. A more sophisticated filter would delay endpoint packets so that they appear to the congested router to have been sent by a slow-starting endpoint or do some sort of traffic editing. An example of traffic editing can be applied in video transmissions. Currently, only the source can adapt the encoding of video transmissions. Video congestion controls could recode video transmissions at the previous uncongested router, thereby providing faster response than waiting for the source to reduce transmission quality.

Finite State Machine Model

The state transition based models [27] represent a network protocol in terms of a finite state machine. The finite state machine is the simplest and most general tool for computing. It has a number of states and it interprets an input symbol and produces an output based on the input and the state the machine currently exists in.

In the previous section, we saw the components that are part of the Active Congestion Control Framework. In this section, we have presented each component as a finite state machine model. The finite state machine model is useful for verifying the correctness of a protocol model specification. The model is decomposed into a set of states and the working of the model is embedded in the transitions between the states. The correctness of the model is verified by validating that for any set of valid inputs to the machine, the Finite State Machine Model generates correct outputs and/or proceeds to valid termination states.

In the finite state machine models, the oval shape represents the state of the machine and the arrows indicate the actions that cause the transitions from one state to another.

Service Manager

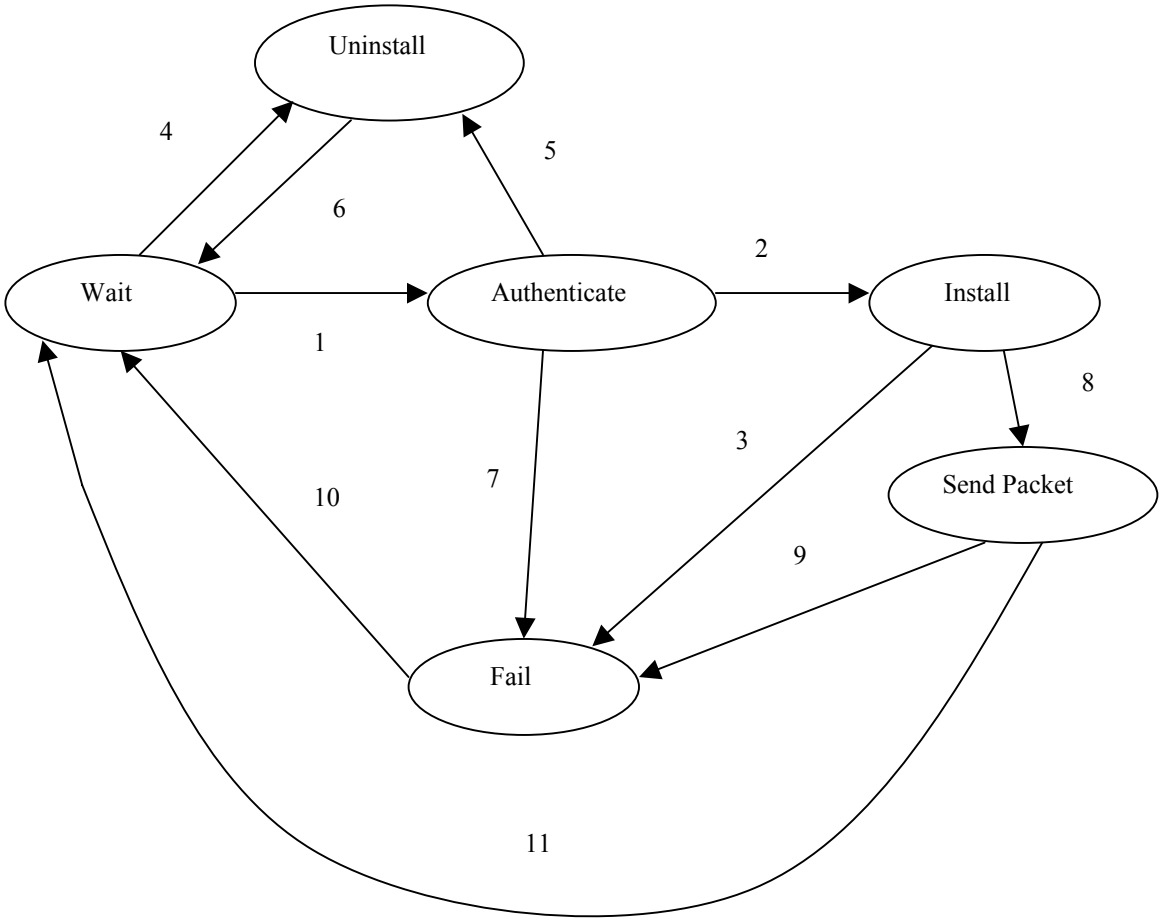


Figure 4: Finite State Machine for Service Manager

State	Explanation
Wait	The Service Manager is waiting for incoming packets.
Authenticate	The Service Manager sends a message that needs to be authenticated to the Authentication Server.
Install	The Service Manager directs the Filter Control Manager to install the filter, by providing the required information.
Uninstall	The Service Manager directs the Filter Control Manager to uninstall the filter. This may either be due to a timeout or an explicit request from a sender regarding changing the filter.
Send	Packets are sent towards the destination.
Fail	Failure occurs in various states, namely, authenticate, install or due to corrupted packets.

Table 1: States in FSM for Service Manager

Transition Event	Explanation
1	The Service Manager receives a message.
2	The authentication of the message is successful.
3	The filter service is preempted because of the request for resources with a higher priority.
4	There is a timeout at the Service Manager, due to inactivity in a particular process.
5	An explicit request is made by a sender to change the type of filter used, and this request is authenticated.
6	The filter is uninstalled, and the resources associated with it are released.
7	Authentication of a message failed.
8	Successful installation of the filter followed by subsequent packets being sent by the Service Manager.
9	Packets not sent due to corrupted bits.
10	Packet was not sent (Failure) and the Service Manager waits for the next request.
11	Packets sent successfully and the Service Manager waits for the next request.

Table 2: Events in the FSM for Service Manager

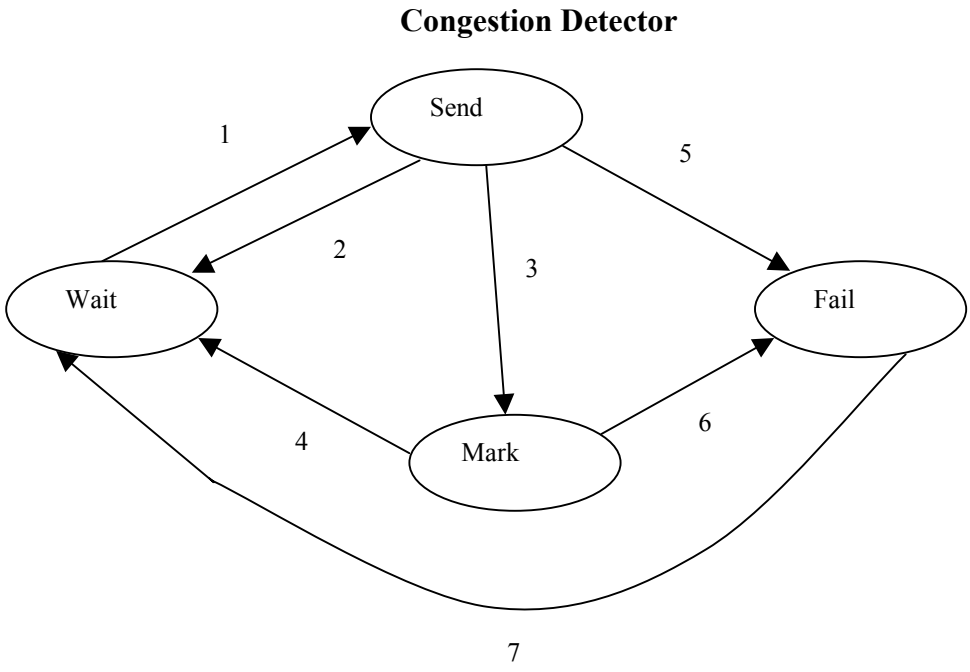


Figure 5: Finite State Machine for Congestion Detector

State	Explanation
Wait	The Congestion Detector is waiting for incoming packets.
Send	The Congestion Detector sends packets towards the destination host or to the Congestion Controller.
Mark	The packet is marked with a probability depending on the Active Queue management algorithm and sent to the Congestion Controller.
Fail	Failure due to corrupted packets.

Table 3: States in FSM for Congestion Detector

Transition Event	Explanation
1	The packet arrives.
2	The packet sent successfully. (didn't cross the minimum threshold level)
3	The packet is above the minimum threshold, and hence, can be marked depending on the probability and also depending on whether it is above the maximum threshold or not.
4	The packet is sent to the congestion controller.
5	The packet couldn't be sent to the destination.
6	The packet couldn't be sent to the Congestion Controller.
7	Failure to send packets due to corruption of bits.

Table 4: Transition Events in FSM for Congestion Detector

Congestion Controller

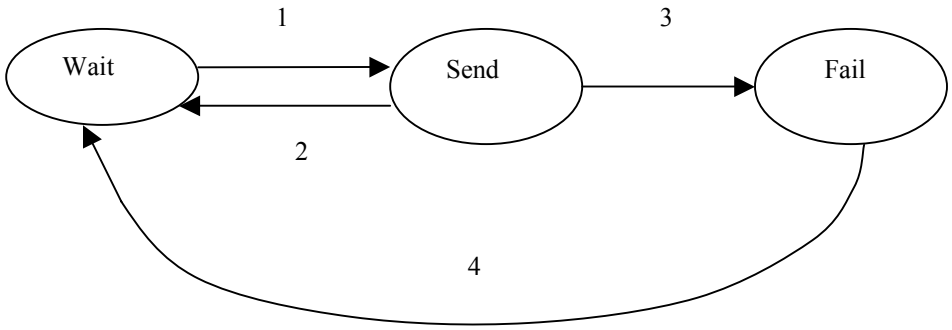


Figure 6: Finite State Machine for Congestion Controller

State	Explanation
Wait	The Congestion Controller is waiting for incoming packets.
Send	The Congestion Controller sends packets to the Correction Filter, through the Filter Control Manager.
Fail	Failure due to corrupted packets.

Table 5: States in FSM for Congestion Controller

Transition Event	Explanation
1	The packet arrives.
2	The Congestion Controller sends the packet to the filter and another packet indicating the current state to the source.
3	The packet couldn't be sent to the filter.
4	Failure to send packets due to corruption of bits.

Table 6: Transition Events in FSM for Congestion Controller

Filter Control Manager

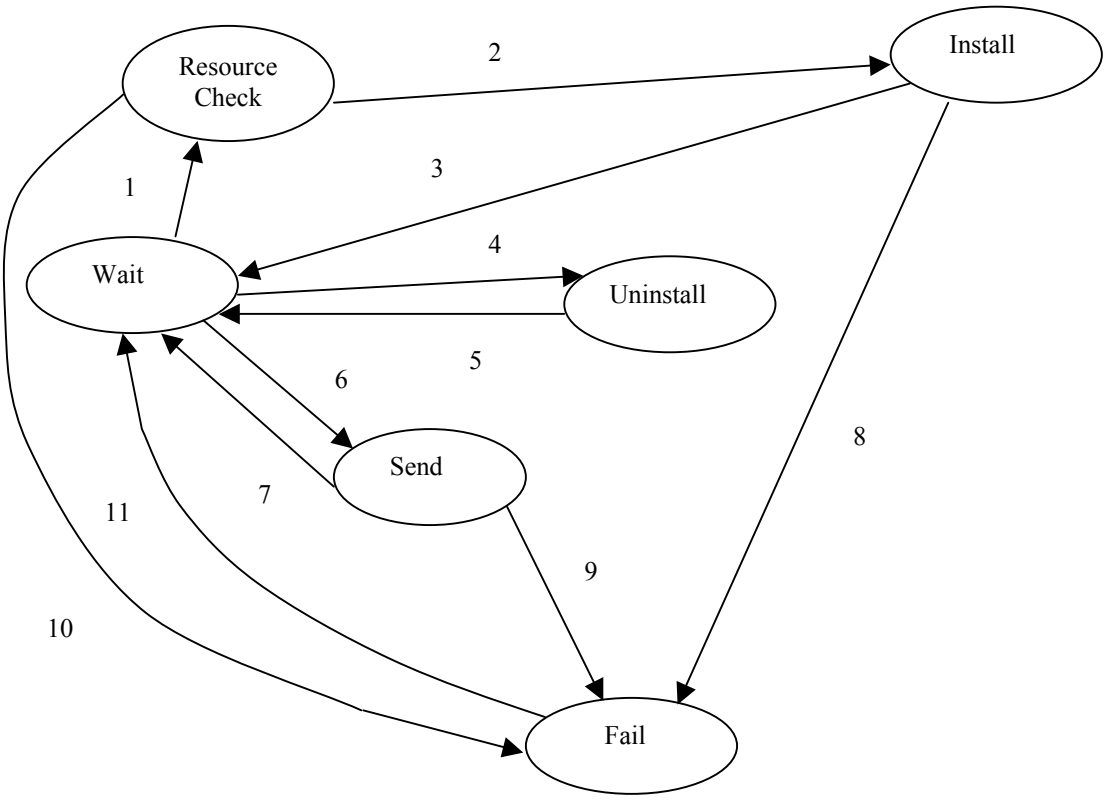


Figure 7: Finite State Machine for Filter Control Manager

State	Explanation
Wait	The Filter Control Manager is waiting for incoming packets.
Resource Check	The Filter Control Manager (FCM) contacts the Resource Allocation Manager to make sure that there is sufficient resource for the request.
Install	The Filter Control Manager installs a new filter, by providing the required information.
Uninstall	The Filter Control Manager uninstalls a filter. This may either be due to a timeout or an explicit request from a sender regarding changing the filter.
Send	Packets are sent through the Correction filter installed.
Fail	Failure occurs due to installation failure or due to corrupted packets.

Table 7: States in FSM for Filter Control Manager

Transition Event	Explanation
1	A request for filter installation from Service Manager received.
2	The Resource allocation Manager approved the request.
3	The filter is installed successfully.
4	A request for filter uninstallation is made either due to inactivity in a particular process or due to explicit filter change request from sender.
5	The filter is uninstalled, and the resources associated with it are released.
6	The packet is to be sent through the existing filter.
7	The packet is successfully sent through the filter.
8	Installation of the filter failed.
9	Packet is corrupted and hence couldn't be sent through the filter.
10	Resources are not available for the requested Filter Control.
11	Failure to send the packet and back to the wait state.

Table 8: Transition Events in FSM for Filter Control Manager

Authentication Server

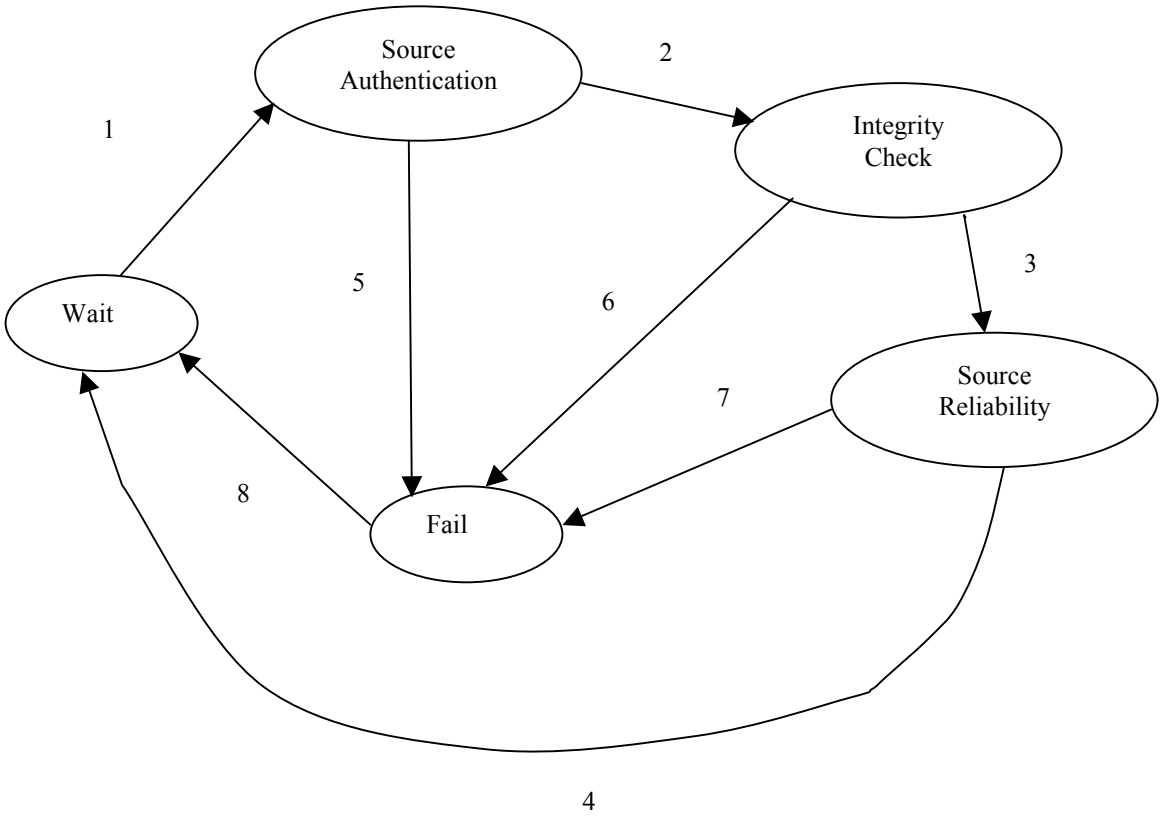


Figure 8: Finite State Machine for Authentication Server

State	Explanation
Wait	Waits for the Authentication Request.
Source Authentication	A check is made to verify whether the source of the message is indeed what is mentioned in the packet.
Integrity Check	The integrity of the message is checked, to make sure that the message is not modified while in transit.
Source Reliability	A check is made to find if the source is reliable.
Fail	Failure occurs due to failure in source authentication, integrity check or source reliability.

Table 9: States in FSM for Authentication Server

Transition Event	Explanation
1	A request for authentication is received.
2	Source Authentication is successful.
3	Integrity check is successful.
4	The source is found to be reliable.(trusted source)
5	Source Authentication has failed.
6	Integrity check has failed.
7	The source is found to be unreliable. This might be either because the source is unknown or because it had misbehaved in a previous instance.
8	A message is sent to the component that made the authentication request, with a failure status.

Table 10: Transition Events in FSM for Authentication Server

Features of Active Congestion Control scheme

Dynamism: Network traffic and configurations vary continuously. The optimal operating point is therefore a continuously moving target. This scheme dynamically adjusts according to the traffic conditions, based on the state information.

Minimum oscillation: The oscillations in the window sizes are reduced, since the active router avoids roundtrip delays.

Convergence: Since there are minimum oscillations, the system reaches convergence faster.

Robustness: The active router takes care of widely varying service-time distributions.

Summary

In this chapter, we have discussed the various issues in router policies and congestion control/avoidance and have proposed a model for active congestion control. We have also looked at the terminology used in the framework and the various components in detail along with the finite state machine model of each component. Further, the states and the transition events in each finite state machine model have been explained. In the next chapter, we look at the spin model, with the message parameters in the model along with the verification results.

Chapter 5: Specification and Verification of the Active Congestion Control Framework

In the previous chapter, we have looked at the Framework for Active Congestion Control and have also looked at how the individual modules interact with each other. This chapter deals with the further details about the modules, including the message parameters and the terminology used.

Message Parameters

There are different types of messages exchanged between the different modules in our proposed framework. These messages have several parameters, which are discussed below:

Packet type

This carries information about the type of packet. This can be data packets, which just carry data, and the state information or the acknowledgement packets from the receiver to the sender or the control packets. There are several service request packets, like the filter installation request packets, filter update packets, resource allocation packets etc.

Process ID

This is used to differentiate between the various processes from the same source, but which might use different filters.

Authentication ID

This field, carries information required to authenticate the sender. In Certificate Authority based authentication, this carries the certificate. In the case of Key Distribution Center based authentication, it is the ticket. This field is used by a component when communicating with the authentication manager. The authentication manager would then decide whether the message is valid or not and send the response to the receiver.

Sequence Numbers

Sequence numbers are provided in the packets, which are used for flow control.

State Information

This carries the state information of the sender, which in our case, is the sending window size.

Filter Setup Priority

Each source is associated with a priority. When a source requests a particular type of filter, if the resources available are insufficient for this filter, this setup priority is compared with the holding priority of the currently active filter. If this setup priority is higher, the current filtering operation is preempted to free the required resources. The resource manager takes care of this.

Filter Holding Priority

Each source is also associated with a holding priority. As explained before, if the available resources are not sufficient for a particular type of filter, the setup priority is compared with the holding priority.

Source/Destination Address

These fields indicate the source and destination address. The address could be an IPv4 or an IPv6 address. The address should be encoded as a TLV, where the type and the length field indicate the type of address.

Component ID

Each of the components in the model has a component ID, which is used for unique identification.

Failure Information

While processing a message, if an error is encountered, a notification message is sent to the sender. Some additional information is provided along with the notification message, such as, information about the resources used or authentication failure etc.

Miscellaneous Attributes

In addition to the fields mentioned above, there is additional information that needs to be sent. Some of these situations are mentioned below:

- When the value of certain parameters received in a message are not acceptable, the receiver needs a mechanism to negotiate the value of those parameters with the sender
- When the sender needs to modify something, say the type of filter installed for a particular session; it needs to convey this message to the Active Router.

Packet Header

Every packet has the message type, message length, source / destination address, the process ID and the authentication Info in addition to the message being sent. These fields comprise the Common Packet Header that accompanies all the messages.

Message Encoding

The various fields mentioned above are encoded as Type/Length/Values (TLVs) as compared to static encoding. There are several advantages of using TLV encoding over the usual static encoding of the packet fields. New TLVs can be added or removed as and when needed by the service. Thus, TLV encoding is very suitable for the deployment of composable services. But TLV has the overhead of increasing the processing time. Yet, it is better that the conventional static encoding, as it is more flexible and hence more efficient when used in protocols that are expected to change. Further, static encoding also has the problem of requiring a specific format, and bit padding, in cases, which is not present in TLVs.

Common Error Conditions

Here, the various conditions that might lead to failure are discussed. When a packet is processed, there might be failure due to

(1) Resource Unavailable
(2) Authentication Failure
(3) Filter Installation Failure
(4) Errors due to corrupted packets
(5) Timeout errors
(6) Service Preemption Errors

Formal Model

The framework is modeled in SPIN and its behavior is verified. The components interact with each other through the information and control messages passed through the communication channels.

The following properties are assumed to hold true in the system:

A1 : No component changes the value of a variable while it is in use by another component.

A2 : The execution of the components in a framework is sequential. This prevents simultaneous execution of two components in a framework on the same node.

A3 : The components or entities communicate through duplex channels.

A4 : Every component may define additional constraints on the state of the environment.

A5 : Read-write access is restricted to certain variables. One such example is the Authentication ID.

A6 : The Authentication Server is trusted by all components.

A1 and A2 avoid race conditions to enable interactions between components to be well defined. Assumption A3 enables asynchronous communication between the components. A4 states that a component verifies its assumptions about the environment by imposing constraints.

Now we define the correctness of the composition by defining the following requirements:

C1 : In every state, all constraints set on the current state by components are satisfied.

C2 : The system has no states that are unreachable.

C3 : Every component either eventually relinquishes control or makes progression.

The first condition prevents components from changing the system environment in an irresponsible manner. The last two conditions assert that a composition is complete and structurally sound, that is, it has exactly all the necessary components.

System Specification

Verification in SPIN involves defining the model in its input language, Promela. The Promela program is fed to the SPIN model checker that tests the correctness, completeness and consistency of the composition. The graphical interface, Xspin is used for verification.

In this section, we outline the salient features of our specification. We discuss how the system is modeled, how the components are defined and how their properties are specified for later verification. We describe the verification techniques used by SPIN to test the models and describe the verification of the various properties of the model.

System Model

The system model consists of the specifications of the components that comprise the active congestion control framework. The environment provides the framework with possible inputs (hosts, in our case) to test the behavior of the framework and determine its correctness. Events such as resource unavailable, authentication failure, corrupted packets and timeout errors have been modeled in our specification.

Defining the Components in the Active Congestion Control Framework

Every component is defined as a separate entity in Promela. A component's interface and behavior are specified using Promela language constructs. Promela does not provide the notion of objects; therefore we model each component in SPIN as a process, defined by the keyword proctype.

Symbolic Constants

The message types are declared as symbolic constants using mtype. The message types used in our model are

```
mtype = {   snd_data,               // Send Data
            snd_ack,                // Send Acknowledgement
            auth_req,               // Authorization Request
            auth_rep,               // Authorization Reply
            filter_install_req,     // Filter Installation Request
            filter_install_rep,     // Filter Installation Reply
            filter_uninstall_req,   // Filter Uninstallation Request
            filter_uninstall_rep,   // Filter Uninstallation Reply
            filter_update_req,      // Filter Update Request
            filter_update_rep,      // Filter Update Reply
            res_alloc_req,          // Resource Allocation Request
            res_alloc_rep,          // Resource Allocation Reply
```

```
decr_snd_wnd,                    // Decrement Send Window
misc,                            // Miscellaneous message
unidentifiable                   // Unidentifiable message
}       ;
```

Only one mtype-definition is allowed which must be global and at most 256 symbolic constants can be declared. The advantage of mtype over #defines is that the former type of symbolic constants is recognized by SPIN and during simulations the symbolic names are used instead of the values they represent [28, Rob Gerth].

Structures

User-defined data types are supported through typedef definitions. We have defined several data types, such as

```
typedef filter_update_pkt {
    mtype msgtype;
    byte flt_id;
    byte src_id;
    byte auth_id;
    byte setup_prio;
    byte hold_prio;
    byte result;
    } ;
```

Variable declarations are done using this type definition as filter_update_pkt flt_pkt; The elements of this structure are accessed as in C, like, flt_pkt.flt_id.

Processes

The process declaration has the form

```
Proctype process_name (parameters)
```

{

Statements

}

The processes are instantiated by a run operation:

Run process_name(parameter values)

Message Channels

The various processes communicate with each other through channels. One such example is

Chan host_svcmgr[2] = [0] of { data_pkt };

Here host_svcmgr is an array of channels; each channel is synchronous, that is, sends and receives must synchronize as no messages can be stored.

Atomic Statement

The atomic statements are executed in one indivisible step; i.e. without interleaved execution of other processes. An extract from our model, where we have defined an atomic statement, and hence is executed without interleaved execution of other processes is:

```
atomic {
        auth_svr[0]!auth_pkt;
        auth_svr[1]?auth_pkt;
        if
          :: auth_pkt.result == 1;
            authent_fail = 0;
            flt_pkt.auth_id = svc_id;
                to_fcm!flt_pkt;
                from_fcm?flt_pkt;
```

```
                    if
                    :: flt_pkt.result == 1;
                            flt_installed = 1;
                    :: skip;
                    fi;
            :: authent_fail = 1;
            fi;
    }
```

An atomic statement is enabled if its first statement is. During its execution, control can only be transferred outside the scope of an atomic statement by an explicit goto or at a point where a statement within its scope becomes blocked. If this statement subsequently becomes enabled again, execution may continue at that point. There is no constraint on what may occur inside the scope, other than that no nested atomic is allowed. In particular, it is possible to jump to any labeled location within the scope of an atomic.

Non-deterministic selection statements

The statement

```
If
:: statements
:: statements
fi
```

selects one among its options, each of which starts with a :: and executes it. An option is selected if its first statement is enabled. A selection blocks until there is at least one selectable branch. If more than one option is selectable, one will be selected at random. This non-determinism is used to model several features in our model, such as error messages. One such example is

```
if
        :: flt_pkt.msgtype = filter_install_req;
        :: flt_pkt.msgtype = filter_uninstall_req;
        :: flt_pkt.msgtype = filter_update_req;
        :: flt_pkt.msgtype = unidentifiable;
        fi;
```

Since all of the above statements following the :: are assignment statements, one of them is chosen in random and the message type is assigned.

Repetition Statements

The repetition statements are modeled as

```
do
::  statements
:: statements
do;
```

These statements are similar to a selection statement, except that the statement is executed repeatedly, until control is explicitly transferred to outside the statement by a goto or break. A break will terminate the innermost repetition statement in which it is executed and cannot be used outside a repetition. These statements are used in our model to represent the functionality of many processes, such as service manager and hosts.

Temporal Claims

Temporal claims are defined by Promela never claims and are used to detect behaviors that are considered undesirable or illegal.

A simple example of the never claim in the model is

```
never {
        do
        ::  authent_failure->break;
        :: skip;
        od;
        do
        :: flt_installed;
        od;
}
```

When checking for state properties, the verifier will complain if there is an execution that ends in a state in which the never claim has terminated, i.e., has reached the closing braces of its body. When checking for acceptance cycles, the verifier will complain if there is an execution that visits infinitely often an acceptance state. Thus, a temporal claim can detect illegal infinite behavior by labeling some statements in the never claim with an acceptance label.

A never claim is intended to monitor every execution step in the rest of the system for illegal behavior and for this reason, it executes in lock-step. Such illegal behavior is detected if the never claim matches along a computation. If a claim blocks (because no statement in its body is enabled) but it is not at its closing braces, then there is no need to explore this computation any further because it cannot lead to a violation.

Verification of Properties using SPIN

SPIN is used to perform on-the-fly verification of the Promela specification generated for the system. SPIN enables verification of liveness and safety properties as well as temporal properties of the model. In SPIN, the verification of these two classes of properties is performed separately.

Verification of safety properties involves checking for correctness and completeness of the composition. This implies checking for any assertion violations and testing for any unreachable code or unspecified receptions. Verification of liveness properties involves ensuring that the system does not enter into any deadlock or livelock, or non-progress execution cycles. Temporal properties can be defined and verified to ascertain specific behavioral properties of the model. These linear time temporal constraints are verified using never-claims.

Correctness and Completeness Verification

Correctness of a composition requires that the composition be structurally sound, that is, the framework model meets requirements C1-C3. This ensures that all component interfaces are invoked correctly, there is no violation of read/write sequence and all constraints set by the components are satisfied.

Checking the syntax of the specification enables us to catch any incorrect calls to component interfaces. Testing the model for safety properties automatically flags any violation of the write/read sequence for packet variables. Component constraints are written in the form of assert statements. Therefore, any violations of the constraints placed by the module are also flagged while checking for safety properties. Checks on the safety properties of the system describe what is allowed to happen. However, just because safety properties hold, that is, nothing bad happens, it does not guarantee that anything does happen. Liveness restricts the long-term behavior of the system by specifying what must eventually happen. Progress must be guaranteed, that is, there are no deadlocks or livelocks. None of the flows should end in a state from which there is no progress. In our framework, every component either progresses to completion or to an explicitly marked acceptance cycle. The SPIN verifier catches the halted progress conditions by checking the model for non-progress cycles. The SPIN verifier also checks to see if the composite has unreachable code, that is, states that the system never gets into.

Verification of Temporal Properties

SPIN also checks correctness properties expressed in linear temporal logic (LTL). Temporal claims are properties of the type " every state in which property P is true is followed by a state in which property Q is true". There are two interpretations of the term "followed by", depending on whether the state Q follows P immediately or eventually after. SPIN makes no assumptions of the relative timing of process executions. Therefore, the interpretation in SPIN is that Q "eventually" follows P.
LTL properties are expressed as never claims to the system, i.e., they are used to formalize system behaviors that are claimed to be impossible. We can utilize LTL formulae to check correctness of temporal properties and consistency of the framework. Checking the correctness of temporal properties of the framework enables us to validate its operation. We have already seen an example of a temporal claim in the section under never claims.

Verification Results

The verification was carried out for a specification model with a simple setup where there are three hosts and a router. Further, the setup was modified to include other cases, so that the model is extensively tested.

Increasing the number of hosts and routers

The number of hosts and routers were increased in steps and the specified properties were found to hold true.

Interleaving trusting and non-trusting hosts

As mentioned earlier, the hosts can either trust the service manager or not trust the Service Manager. Both kinds of hosts were included in the model and the specified properties were verified.

Changing the filter

The default filter, where the packets marked by the RED router are just dropped was used and then a more sophisticated filter, which dropped only the packets of lower priority was used. This also helped us to verify the composability of the components.

Active and non-active hosts

Both active and non-active hosts were used in the model and it was found that the model properties still held true and the system was verified. Further, only non-active hosts were modeled and the framework still functioned as expected, exhibiting compatibility with the existing network.

Active And Non-active Routers

The model was verified after replacing the active RED router with a tail-drop router.

Observations

The following observations were made from the various test cases that were verified:

- The state space and memory used for verification increases as the number of components in the system increases, as the complexity of the system increases.

- In the absence of active hosts, the framework functioned normally and the properties held true, thereby exhibiting compatibility.

- The framework has functionally separate components, and hence changing the filters was easy.

Summary

In this chapter, we defined the concepts of completeness, correctness and consistency for the active congestion control framework. We also defined the message parameters and the terminology used. The specification and verification of the system is described in detail, and the chapter ends with the verification results and observations.

Chapter 6 : Summary and Future Work

Active networking provides a new paradigm of networking in which users are able to create and inject custom services and protocols in the network. This thesis presents a model for actively controlling congestion in the network. The properties of the active congestion control framework model for active networks are identified by studying the limitations of current models and analyzing requirements of the framework. This thesis then describes the importance of verification in a Framework Model.

This thesis identifies the various components in the proposed Active Congestion Control framework. We then describe the Finite State Machine Models for each of the components. The Finite State Machine explains the interactions with the various components and the events that follow the interactions. We then look at the various message types used in our model and the various fields described in the model.

This thesis also describes the specification aspects of the proposed model. The components are specified using the SPIN verification system. The functionality of individual components are described as properties of the component using SPIN's input language called Promela. Various test cases are considered for the verification process. The SPIN verifier is used to check the correctness and completeness of the composed specification and the observations inferred from the results are discussed.

The proposed framework is found to function and satisfy the specifications mentioned. We have just used the RED and tail-drop routers in our model. Other router models could be used and the framework could be tested. An implementation based on this framework would be a feasible future work.

Bibliography

[1] David.L.Tennenhouse and David.J.Wetherall,"Towards an Active Network Architecture". In Computer Communication Review,1996.

[2] David.L.Tennenhouse, Jonathan M.Smith, W.David Sincoskie, David J.Wetherall, Gary J. Minden, "A Survey of Active Network Research". In IEEE Communications Magazine, Vol. 35, No.1, pp. 80-86, January 1997.

[3] E.Amir, W.McCanne, and H.Zhang, "An application level video gateway". In ACM Multimedia '95, 1995.

[4] Greenwald, M., et al., "Designing an Academic Firewall: Policy, Practice and Experience with SURF". In Proc. of the 1996 Symp. on Network and Distributed Systems Security, 1996.

[5] G.R.Wright and W.R.Stevens, "TCP/IP Illustrated", Volume 2.

[6] Van Jacobson and Michael J.Karels, "Congestion Avoidance and Control".

[7] Matthew Mathis and Jamshid Mahdavi, " Forward Acknowledgement: Refining TCP Congestion Control.

[8] Sally Floyd , " TCP and Explicit Congestion Notification", ACM Computer Communication Review, Vol. 24, pp. 8-23, Oct. 1995.

[9] A. Mankin, "Random Drop Congestion Processing". In Proceedings of the 1990 SIGCOMM Conference, pp. 1-29. September 1990.

[10] Floyd, S., and Jacobson, V., "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, Vol. 1 N.4, August 1993, p. 397-413.

[11] Hong Peng, Sofiene Tahar and Ferhat Khendek, " Comparison of SPIN and VIS for Protocol Verification".

[12] G.J.Holzmann, "Design and Validation of Computer Protocols", Englewood Cliffs, J.J, Prentice Hall, 1991.

[13] E. Clarke and J.Wing, "Formal Methods : State of the Art and Future Directions ", ACM Computing Surveys, Vol.28, No.4, pp 626-643, December 1996.

[14] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation", IEEE Trans. on Computers C-35,8, 1986.

[15] R. P. Kurshan, "The Complexity of Verification", In Proc. 26th ACM Symposium on Theory of Computing, pp.365-371, Montreal 1994.

[16] D. Peled, "Combining partial order reductions with on-the-fly model-checking", Proc of the 6th International Conference on Computer Aided Verification 1994, pp.377-390, Stanford CA.

[17] W. Elseaidy, R. Cleaveland and J.Baugh, "Modeling and verifying active structural control systems", 1996, In Proc of the 1994 Real-Time Systems Symposium.

[18] Gerald Holzmann , " The Model Checker SPIN", IEE Transactions on Software Engineering, vol.23, no.5, May 1997.

[19] Gerald.J. Holzmann and Peled, " An Improvement in Formal Verification", Proc. Seventh FORTE Conf. Formal Description Techniques, pp. 177-194, Bern, Switzerland, Oct. 1994.

[20] Gerald Holzmann, " An Analysis of Bit-State Hashing ", Proc. IFIP/WG6, Symposium on Protocol Specification, Testing and Verification, pp.301-314, Warsaw, Poland, June 1995.

[21] Ted Faber, " ACC: Active Congestion Control", IEEE Network, IEEE, May/June 1998, pp. 61-65.

[22] Samrat Bhattacharjee Kenneth L. Calvert Ellen W. Zegura, "On Active Networking and Congestion", 1996.

[23] Sally Floyd and Kevin Fall, "Promoting the Use of End-to-End Congestion Control in the Internet, IEEE/ACM Transactions on Networking, May, 1999.

[24] Bernhard Suter, T.V.Lakshman, Dimitrios Stiliadis, Abhijit Choudhury, "Efficient Active queue Management for Internet Routers"

[25] Raj Jain, K.K.Ramakrishnan, Dah-Ming Chiu, " Congestion Avoidance in Computer Networks with a Connectionless Network Layer".

[26] E. W. Zegura, "Composable Services for Active Networks", Active Network Composable Services Group Working Draft, May 1998.

[27] G.Bochmann, "Finite State Description of Communication Protocols", Computer Networks, Vol.2, Oct. 1978.

[28] Rob Gerth, " Concise Promela Reference".